Improving the Performance of the Contextual Spaces Re-Ranking Algorithm on Heterogeneous Systems

Flávia Pisani[1], Daniel C. G. Pedronette[2], Ricardo da S. Torres[1], Edson Borin[1]

[1]Institute of Computing (IC) – University of Campinas (UNICAMP)
Campinas, São Paulo, Brazil

[2]Institute of Geosciences and Exact Sciences (IGCE) – São Paulo State University (UNESP)
Rio Claro, São Paulo, Brazil

{fpisani,rtorres,edson}@ic.unicamp.br, daniel@rc.unesp.br

Abstract. *Re-ranking algorithms have been proposed to improve the effectiveness of Content-Based Image Retrieval (CBIR) systems by exploiting contextual information encoded in distance measures and ranked lists. In this paper, we show how we improved the efficiency of one of these algorithms, called Contextual Spaces Re-Ranking. We propose a modification to the algorithm that reduces its execution time by* $1.6\times$ *on average and improves its accuracy in most of our test cases. We also parallelized the implementation with OpenCL to use the CPU and GPU of an Accelerated Processing Unit (APU). Employing these devices to run different parts of the code resulted in speedups that range from* $3.3\times$ *to* $4.2\times$ *in comparison with the total execution time of the serial version.*

1. Introduction

Due to a reduction in price of storage devices and the new technological advances that are being made in the field of data acquisition and sharing, we can observe a considerable increase in the size of image collections. As a result of that, the adoption of search systems becomes very important for users to be able to find images in these huge collections. Widespread retrieval approaches, such as the ones based on keywords and textual metadata, face serious challenges caused by the inherent difficulty in describing an image in words [Datta et al. 2008]. Furthermore, textual image description is an intrinsically time-consuming and laborious task, and it also depends on the subjective, and usually inconsistent, evaluation of annotators.

Content-Based Image Retrieval (CBIR) is a technology that mitigates this problem by providing automatic mechanisms for searching based on an image's visual properties (e.g., color, shape, and texture). Given a query image, a CBIR system intends to retrieve similar items from the collection by using one or more content descriptors, which encode the visual properties of the images. A CBIR system ranks the results by decreasing order of similarity, and since users consider mostly top-ranked images, it is imperative that the rank be as accurate as possible.

In recent years, several successful attempts to increase the effectiveness (quality of results) of CBIR systems have been performed [Datta et al. 2008, Pedronette and da S. Torres 2011, Yang and Latecki 2011]. In particular, re-ranking

methods have been used to improve the effectiveness of CBIR systems by exploiting contextual information encoded in similarity scores and ranked lists. These methods are, on the other hand, very costly as they are based on comparing collection images multiple times. In a real-world scenario, CBIR systems require both good effectiveness and efficiency (response time), so re-ranking methods must be improved.

Central Processing Units (CPUs) no longer have just one core and Graphics Processing Units (GPUs) are now being used as general-purpose processors (GPGPUs) since they have evolved into massive parallel architectures capable of executing hundreds of operations per cycle [Pedronette et al. 2012]. These devices have been successfully used to accelerate re-ranking [Pedronette et al. 2012, Pedronette et al. 2013] and retrieval [Teodoro et al. 2014] systems, obtaining good speedups.

Therefore, alternatives that increase performance through parallelization seem like a possible fit for the Contextual Spaces Re-Ranking algorithm [Pedronette and da S. Torres 2011], which we discuss in this paper. Another approach that can be pursued is analyzing the compromises between accuracy and performance that come out of modifying existing algorithms, since this can lead to removing demanding work from the execution.

Our solution exploits the use of parallelization to speed up the more costly steps of a modified version of the Contextual Spaces Re-Ranking algorithm. The experiments made on an Accelerated Processing Unit (APU) show that this approach improves the total execution time by $1.6\times$ on average, while also increasing the Mean Average Precision (MAP) score in most of our test cases. To the best of our knowledge, our research group has the only efforts to parallelize the Contextual Spaces Re-Ranking algorithm.

The remainder of this paper is organized as follows: Section 2 presents related work. Section 3 introduces the Contextual Spaces Re-Ranking algorithm and the modifications we propose. Section 4 describes our APU-based implementation. Our experimental results are discussed in Section 5. Finally, Section 6 draws our conclusions.

2. Related Work

In image retrieval applications, images are commonly represented by feature vectors in an attempt to represent their visual properties. The comparison between two images is performed using pairwise analysis, e.g., computing the Euclidean distance between their correspondent feature vectors. Nonetheless, the pairwise distances ignore the relationships among images and the context in which the query is processed, failing to retrieve relevant results in many situations.

Recent studies [Pedronette and da S. Torres 2011, Yang and Latecki 2011] have focused on post-processing methods. Such approaches aim at replacing pairwise distances by more global measures, capable of considering the overall dataset structure. These methods present an important advantage, as they do not require any user intervention. Since labeling is often a laborious and time-consuming task, these unsupervised approaches are a very attractive solution for improving the effectiveness of CBIR systems.

However, several of these methods require high computational efforts, mainly due to matrix operations [Yang et al. 2008] or graph procedures [Wang et al. 2011], and their evaluation commonly considers only effectiveness, ignoring efficiency aspects that are

also indispensable in real-word applications. Two alternatives have recently emerged as possible solutions: rank-based approaches, in which the information of top-k retrieved images is exploited, reducing the required computation; and the use of parallel and heterogeneous architectures, which are popular for accelerating different image applications.

Graphics Processing Units (GPUs) are now viewed as inexpensive co-processors suited for many applications beyond computer graphics [Steele and Cochran 2007], leading to General Purpose GPU (GPGPU) approaches. Emerging heterogeneous devices like Accelerated Processing Units (APUs) are also contributing to disseminate the use of parallel systems. GPUs are being used in many different areas, such as general image retrieval, remote sensing [Sevilla et al. 2014], and medical applications [Ferreira et al. 2014].

This paper discusses a modified version of the Contextual Spaces Re-Ranking algorithm, which exploits information from rankings and distances for improving the effectiveness of image retrieval systems. There are very few studies about the efficient image re-ranking computation on parallel architectures [Pedronette et al. 2012, Pedronette et al. 2013]. Therefore, the efficiency evaluation considering APU devices and heterogeneous computing is also a contribution of this work.

3. Contextual Spaces Re-Ranking Algorithm

The *Contextual Spaces Re-Ranking* algorithm [Pedronette and da S. Torres 2011] aims to redefine the relationships between collection images by answering the question "*What information can an image's nearest neighbors provide about other items in the collection?*". This way, not only the pairwise correlation between images is analyzed, but also the information contained in the context of the query.

3.1. Problem Definition

Let $\mathcal{C} = \{img_1, img_2, \ldots, img_N\}$ be an image collection and $\mathcal{D}$ an image descriptor that defines a distance function $\rho : \mathcal{C} \times \mathcal{C} \rightarrow \mathbb{R}$, where $\mathbb{R}$ denotes real numbers. Consider $\rho(x, y) \geq 0$ for all (x, y) and $\rho(x, y) = 0$ if $x = y$. The distance $\rho(img_i, img_j)$ for each pair of images $img_i, img_j \in \mathcal{C}$ can be calculated to obtain an $N \times N$ distance matrix A.

Given a query image img_q, we can compute a ranked list R_q by taking into account the distances in A. This ranked list can be defined as a permutation of the items in $\mathcal{C}$ such that if img_i is closer to the top of the list than img_j, then $\rho(img_q, img_i) \leq \rho(img_q, img_j)$. Considering each image $img_i \in \mathcal{C}$ as a query, we obtain a set $\mathcal{R} = \{R_1, R_2, \ldots, R_N\}$ of ranked lists, which can also be stored in an $N \times N$ matrix. By taking as input the distance matrix A and the set of ranked lists $\mathcal{R}$, the re-ranking algorithm (represented by the function F) computes a new distance matrix $\hat{A} = F(A, \mathcal{R})$.

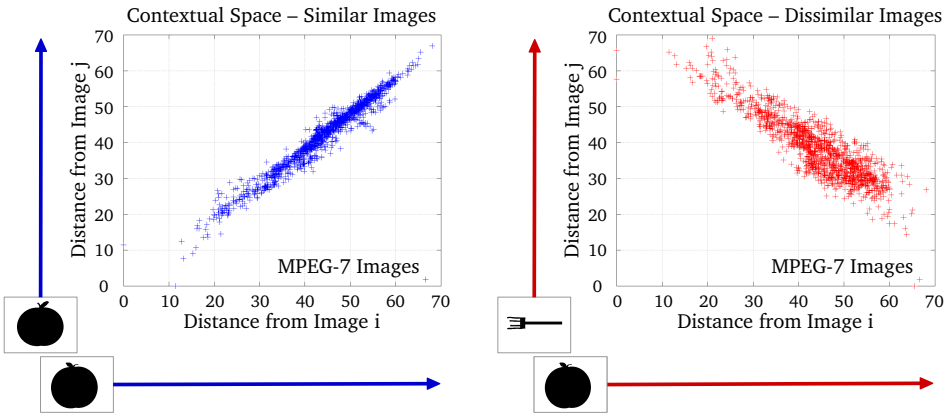
Based on the new distance matrix $\hat{A}$, collection images can be re-ranked, that is, a new set of ranked lists $\hat{\mathcal{R}}$ can be obtained. The re-ranking algorithm based on Contextual Spaces, detailed in this section, consists in an implementation of the function F.

3.2. Contextual Spaces Representation

Given two reference images img_i and img_j, consider a two-dimensional Cartesian space where, for all $img_l \in \mathcal{C}$, the *horizontal axis* represents the values of $\rho(img_i, img_l)$ and the *vertical axis* represents the values of $\rho(img_j, img_l)$. This means that the position of a certain image $img_l \in \mathcal{C}$ is given by the ordered pair $(\rho(img_i, img_l), \rho(img_j, img_l))$.

We take images from the MPEG-7 dataset [Latecki et al. 2000] as an example. By using the CFD shape descriptor [Pedronette and da S. Torres 2010], we can compute the distance between each pair of images in this collection and then build the corresponding space for a given pair of reference images.

Figure 1(a) shows a graphic representation of a space where the reference images are similar. We see that the distances from each reference image to the other images (i.e., the horizontal and vertical coordinates of each point) are similar as well. Figure 1(b), in turn, shows a case where the two reference images are not similar. Note that this time the distances from each reference image to the other images present a negative correlation.



(a) Similar reference images (b) Dissimilar reference images

Figure 1. Graphical representation of two-dimensional spaces constructed with certain pairs of (a) similar and (b) dissimilar reference images.

We can define a *contextual space* as a two-dimensional space considering as reference images an arbitrary image img_i and each one of its K-nearest neighbors $KNN(img_i) = \{img_{j_1}, img_{j_2}, \ldots, img_{j_K}\}$. The Contextual Spaces Re-Ranking algorithm intends to exploit the information of contextual spaces to improve the distances among collection images, which can lead to better rankings.

3.3. Original Contextual Spaces Re-Ranking Algorithm

Algorithm 1 presents the main operations of the Contextual Spaces Re-Ranking algorithm. Two parameters must be set: the initial number of neighbors to be considered, K_s, and the final number of neighbors, K_e. The number of iterations of the algorithm (line 4) is defined as $T = K_e - K_s + 1$.

First (lines 1–3), we initialize the current iteration number (t), the current distance matrix (A_t), and the current number of neighbors to be analyzed (K). In the main loop, for a given pair of images $img_i, img_l \in \mathcal{C}$ (lines 6–7), we consider the contextual spaces that have reference images img_l and each one of img_i's K-nearest neighbors (line 11).

The dimension d_j is computed as a weighted average of the distances from $img_j \in KNN(img_i)$ to img_l (line 12). The term $K - c_k$ (where c_k is a variable declared in line 9) is used as weight so the distances from neighbors closest to the top of the list are considered more relevant. Both dimensions d_i and d_j are then divided proportionally by

Algorithm 1 Contextual Spaces Re-Ranking

Require: Original distance matrix A; set of ranked lists $\mathcal{R}$.
Ensure: Processed distance matrix $\hat{A}$; new set of ranked lists $\hat{\mathcal{R}}$.

1: $t \leftarrow 0$
2: $A_t \leftarrow A$
3: $K \leftarrow K_s$
4: **while** $K \leq K_e$ **do**
5: $\quad A_{t+1} \leftarrow 0$
6: $\quad$ **for all** $img_i \in C$ **do**
7: $\quad\quad$ **for all** $img_l \in C$ **do**
8: $\quad\quad\quad$ {Considering contextual spaces}
9: $\quad\quad\quad c_k \leftarrow 0$
10: $\quad\quad\quad d_j \leftarrow 0$
11: $\quad\quad\quad$ **for all** $img_j \in KNN(img_i)$ **do**
12: $\quad\quad\quad\quad d_j \leftarrow d_j + A_t[j,l] \times (K - c_k)$
13: $\quad\quad\quad\quad c_k \leftarrow c_k + 1$
14: $\quad\quad\quad$ **end for**
15: $\quad\quad\quad$ {Computing distance $\rho(img_i, img_l)$}
16: $\quad\quad\quad d_i \leftarrow A_t[i,l]/K$
17: $\quad\quad\quad d_j \leftarrow d_j/(\frac{K \times (K-1)}{2})$
18: $\quad\quad\quad A_{t+1}[i,l] \leftarrow \sqrt{d_i^2 + d_j^2}$
19: $\quad\quad$ **end for**
20: $\quad$ **end for**
21: $\quad \mathcal{R}_{t+1} \leftarrow performReRanking(A_{t+1})$
22: $\quad K \leftarrow K + 1$
23: $\quad t \leftarrow t + 1$
24: **end while**
25: $\hat{A} \leftarrow A_t$
26: $\hat{\mathcal{R}} \leftarrow \mathcal{R}_t$

the number of neighbors considered in the iteration (lines 16–17) and the new distance between img_i and img_l becomes the Euclidean distance defined by d_i and d_j (line 18).

Once the distances among images are redefined, a new set of ranked lists can be computed (line 21). As we increase the current iteration t (line 23), we can also increment the number of K neighbors considered for constructing the contextual spaces (line 22). The motivation behind this increment relies on the fact that the effectiveness of this approach increases along iterations since new distances are computed and non-relevant images are moved away from the first positions of the ranked lists. A new distance matrix $\hat{A}$ and a new set of ranked lists $\hat{\mathcal{R}}$ are produced as a final result (lines 25–26).

3.4. Proposed Modification

Since the maximum number of neighbors analyzed by Algorithm 1 is a constant much smaller than N (number of collection images), we see that the complexity of the Contextual Spaces Re-Ranking algorithm is determined by the assignment of values to matrices in lines 2–3, 25–26; the loop in line 6; and the re-ranking procedure in line 21.

Assigning values to the matrices and the loop in line 6 both have complexity $O(N^2)$. The procedure in line 21 is defined by N sorting operations on lists of size N. As these lists are almost sorted, we can use the *insertion sort* algorithm to sort them in linear time [Bentley 2000], yielding a complexity of $O(N^2)$ for this operation as well.

The modification we propose consists in moving the re-ranking procedure from inside of the main loop (line 4) to after its last execution. This way, it is now between lines 24 and 25 and is executed only once. This does not affect the complexity of the algorithm, but causes a decrease in the number of re-ranking steps from T to 1.

If we compare a single execution of insertion sort on the original and modified algorithms, the later will have more operations, as it deals with a distance matrix that was altered T times instead of just one. Still, the results presented in Section 5.3.2 show that the change improves the execution time of this step, since fewer operations are executed in total. The effectiveness of the algorithm is also improved for most of the image descriptors used in our tests, as discussed by the evaluation in Section 5.2.

4. Acceleration of the Contextual Spaces Re-Ranking Algorithm

4.1. OpenCL

OpenCL is a new industry standard for task-parallel and data-parallel heterogeneous computing on a variety of modern CPUs, GPUs, and other processors [Stone et al. 2010]. An OpenCL program runs on *computational devices* such as CPUs and GPUs. These usually have *compute units* (processor cores) with one or more single-instruction multiple-data (SIMD) *processing elements* (PE) that execute instructions in lockstep.

A program is divided into *kernels*, which are dynamically compiled OpenCL functions. A kernel's execution on a device is scheduled by a C runtime library. Each SIMD kernel instance is called a *work-item* and executes on a single PE [AMD 2013].

4.2. Parallel Contextual Spaces Algorithm

Some steps of the Contextual Spaces algorithm described in Section 3 have inherent potential for parallelization and can thus be exploited to increase the algorithm's efficiency.

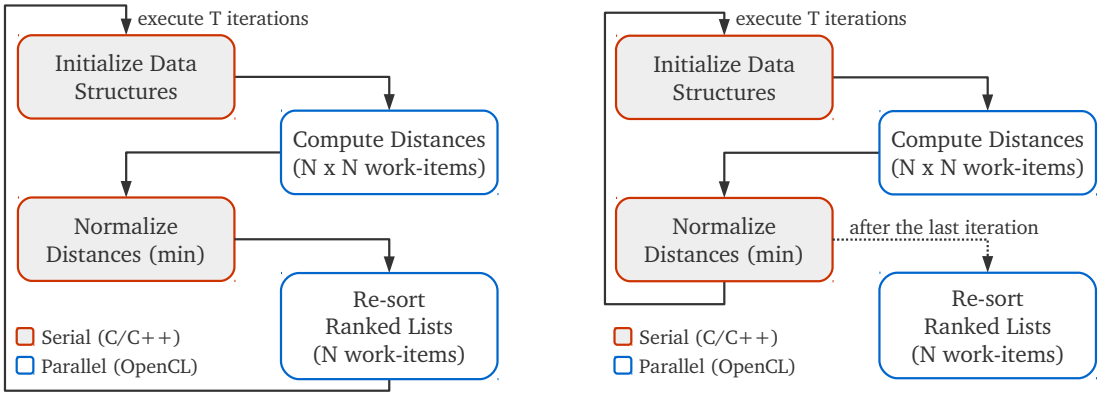
We call lines 6-20 from Algorithm 1 the *Compute Distances* step. We can see that there are no loop-carried dependencies created by the `for` commands in lines 6 and 7, so the computation can be performed concurrently for each pair of images.

Line 21 is the *Re-sort Ranked Lists* step. We observe that in this step, each image's ranked list is independent of other ranked lists, so it becomes clear how we can benefit from processing them concurrently.

To guarantee that the synchronization requirement between steps is met, an OpenCL kernel was created for each of them:

- **Compute Distances:** updates the distance between a pair of collection images considering the contextual spaces formed by the query's K-nearest neighbors. Since this operation can be performed in parallel for all pairs of images in the collection, we have $N \times N$ work-items executing this kernel.
- **Re-sort Ranked Lists:** re-orders an image's ranked list considering the new distances calculated in the previous step. The more similar two images are, the smaller is the distance between them, making more relevant images closer to the top of the list. Since this operation can be performed in parallel for all ranked lists, we have N work-items executing this kernel.

Serial code was added to provide the necessary data input for the kernels. The "Normalize Distances" operation was separated from the Compute Distances step to avoid the use of barriers and we decided to not parallelize it, since it is simple and doing so would only introduce overhead. This configuration created the design depicted in Figure 2(a). After the modification described in Section 3.4, we still have the same parallelization structure, but the execution flow became the one seen in Figure 2(b).



(a) Original (b) Modified

Figure 2. Parallelization of the Contextual Spaces Re-Ranking algorithm.

5. Experimental Evaluation

This section presents the experimental setup and the experiments we carried out to evaluate the effectiveness and efficiency of the parallel implementation described in Section 4.

5.1. Experimental Setup

The APU used in our experiments is the AMD A8-3850, which combines 4 CPU cores with an AMD Radeon HD6550D GPU that has 400 cores. We ran the tests on a Linux 3.3.4-5 Fedora 17 environment with AMD OpenCL SDK 2.8. We compiled the C/C++ serial code with g++ 4.7.2 using the flag "-O3".

The datasets and descriptors we chose are: the MPEG-7 dataset (1,400 images) and shape descriptors SS, BAS, IDSC, CFD, and ASC; the Soccer dataset (280 images) and color descriptors GCH, ACC, and BIC; and the Brodatz dataset (1,776 images) and texture descriptors LBP, CCOM, and LAS. Due to the space constraints of this paper, we use the article that introduces the Contextual Spaces Re-Ranking algorithm [Pedronette and da S. Torres 2011] as a reference for these datasets and descriptors.

The measure we used to evaluate effectiveness is the Mean Average Precision (MAP), geometrically referred to as the area below a *precision* (fraction of retrieved instances that are relevant) $\times$ *recall* (fraction of relevant instances that are retrieved) curve.

For our efficiency analysis, we compared execution times of the serial implementation against the OpenCL kernels running in either the CPU or the GPU. We chose the MPEG-7 dataset and $T = 8$ iterations for our tests. Each test case was executed 20 times and the average running time was computed with corresponding 95% confidence interval.

5.2. Effectiveness of our Parallel Implementation

Table 1 presents effectiveness results in terms of the MAP measure for the parallel implementations of the original and modified Contextual Spaces Re-Ranking.

Table 1. MAP for the Contextual Spaces Re-Ranking tests.

Dataset	Image Desc.	Original Score [%]	Modified Score [%]	Modified vs Original [%]
MPEG-7	SS	40.75	46.08	**+5.33**
	BAS	74.71	77.00	**+2.29**
	IDSC	85.67	88.06	**+2.39**
	CFD	90.00	90.56	**+0.56**
	ASC	90.51	90.69	**+0.18**
Soccer	GCH	32.97	34.32	**+1.35**
	ACC	39.35	40.09	**+0.74**
	BIC	43.07	45.58	**+2.51**
Brodatz	LBP	49.34	49.26	-0.08
	CCOM	61.49	63.28	**+1.79**
	LAS	79.67	79.41	-0.26

The scores achieved by the parallelized version of the original algorithm are very similar to the ones from the serial implementation made when it was first proposed [Pedronette and da S. Torres 2011], but the scores obtained with the modifications present some improvement.

Six out of the eleven tests had an increase in accuracy of at least 1.35% (up to 5.33%). For all MPEG-7 and Soccer tests, the results of the modified version were better than the original. Two cases of the Brodatz tests, however, presented a decrease in the MAP, but no greater than 0.26%.

5.3. Efficiency of our Parallel Implementation

The following performance analysis is divided into two parts: first, we look at the results obtained by parallelizing the original Contextual Spaces Re-Ranking algorithm. Then, we analyze the impact of the algorithm modification. Since the Compute Distances kernel was not modified, the second part focuses on the Re-sort Ranked Lists kernel.

For simplicity, the kernel names in this section are abbreviated to "Dist" (Compute Distances) and "Sort" (Re-sort Ranked Lists). Also, the notation *"k-Impl"* is employed to indicate which implementation of a kernel was used in a certain test case. The value of *"k"* can be either *"s"*, the "Sort" kernel, or *"d"*, the "Distance kernel". The value of *"Impl"* can be *"CPU"*, the OpenCL version executed on the CPU; *"GPU"*, the OpenCL version executed on the GPU; or *"Serial"*, the C/C++ version.

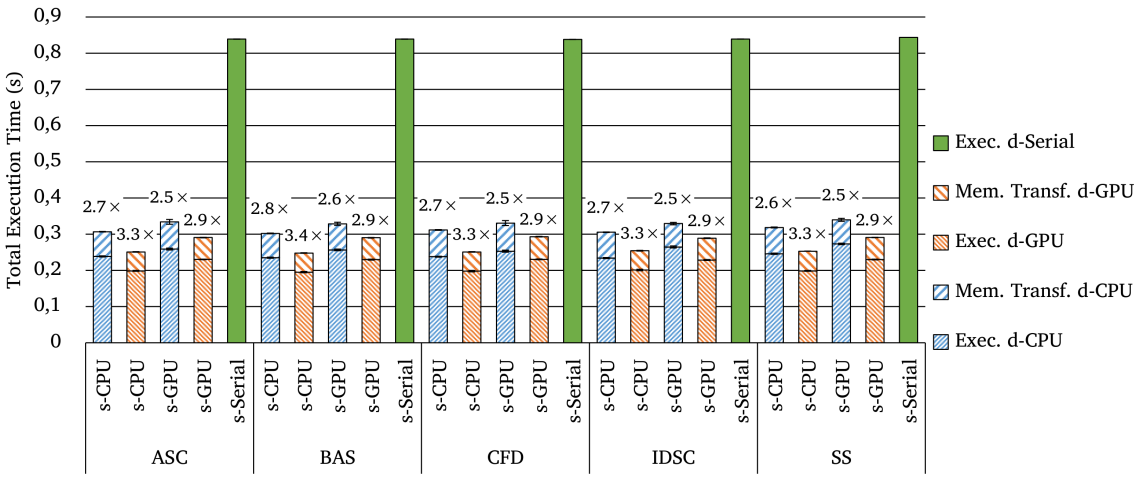
In each graph, it is possible to compare the serial and parallel execution times by calculating the speedup obtained by the parallelization. These speedups are displayed as labels above the bars representing parallel test cases. Since all memory transfer times for the serial executions are equal to zero, this information is not included in the graphs.

The tables bellow the graphs contain the median speedups for each test case considering the descriptors displayed. The underlined kernels in these tables represent the kernel or kernel combination for which the speedups are being analyzed.

5.3.1. Original Algorithm

By using the approach described in Section 4.2, we implemented a parallelized version of the Dist and Sort steps of the original Contextual Spaces Re-Ranking algorithm.

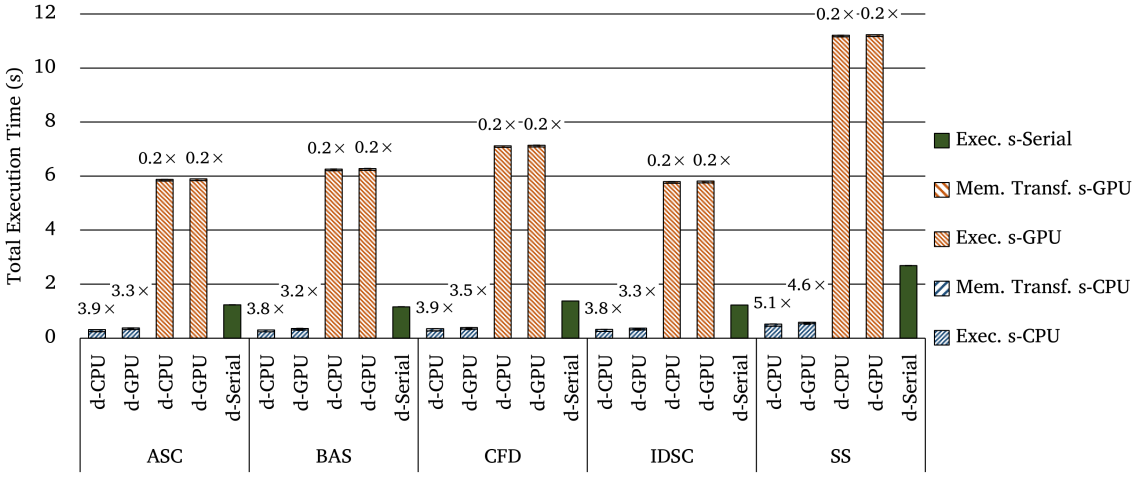
Figure 3 shows the results for the Dist kernel. As expected, the best execution times were obtained when the kernel is running on the GPU (orange bars with downward - i.e. from top left to bottom right - diagonal pattern).



Test Case	d-CPU/s-CPU	d-GPU/s-CPU	d-CPU/s-GPU	d-GPU/s-GPU
Median Speedup	2.7 ×	3.3 ×	2.5 ×	2.9 ×

Figure 3. Comparison of Dist kernel execution times for different descriptors.

Figure 4 displays the results for the Sort kernel. Their most noticeable characteristic is that the s-GPU cases (dark orange bars with downward diagonal pattern) have very high running times, performing considerably worse than the serial version.



Test Case	d-CPU/s-CPU	d-GPU/s-CPU	d-CPU/s-GPU	d-GPU/s-GPU
Median Speedup	3.9 ×	3.3 ×	0.2 ×	0.2 ×

Figure 4. Comparison of Sort kernel execution times for different descriptors.

Although exploring other sorting algorithms is beyond the scope of this paper, we

note that this is due to insertion sort not being a good sorting algorithm for GPUs, so different approaches could significantly improve the results.

Nonetheless, the s-CPU cases (dark blue bars with upward - i.e. from bottom left to top right - diagonal pattern) show that the chosen parallelization approach paid off when it comes to the CPU.

Finally, we analyze the combined running time of both kernels, as shown in Figure 5. To make the graph simpler, we did not include s-GPU cases. We notice that the Sort kernel has great impact on the total execution time, going from 50% (BAS d-CPU/s-CPU) to as high as 76% (SS d-Serial/s-Serial). Therefore, optimizing this kernel is very important for the algorithm's overall performance.

This is specially highlighted by the tests with the SS descriptor. While the median total speedup is $3.3\times$ for both d-CPU/s-CPU and d-GPU/s-CPU, SS presents speedups of $4.2\times$ for the same cases, since its Sort kernel takes longer than the others.

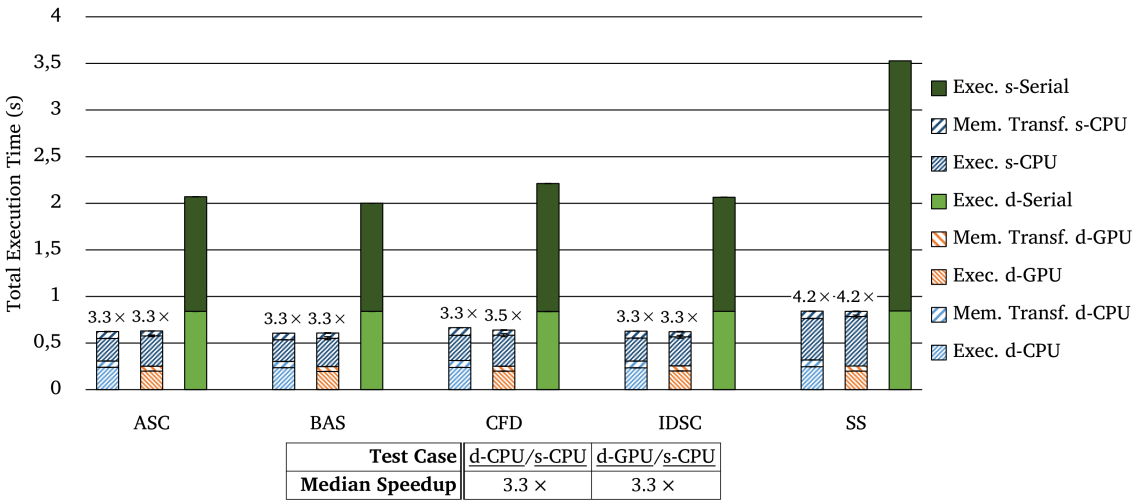
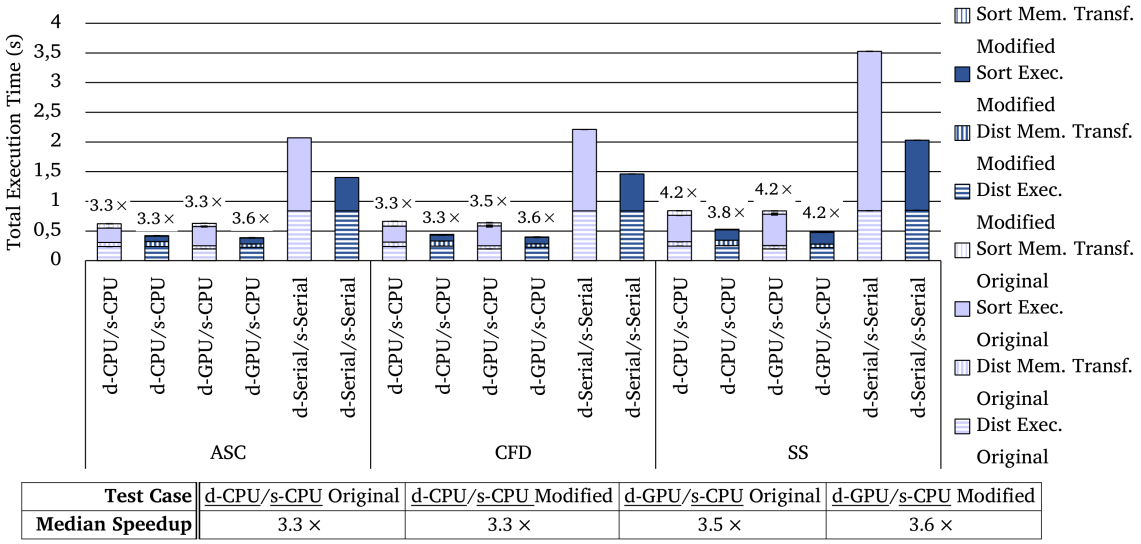


Figure 5. Comparison of total kernel execution times for different descriptors.

5.3.2. Modified Algorithm

As it was argued in Section 5.3.1, optimizing the Sort kernel of the Contextual Spaces Re-Ranking algorithm is important to improve its execution time. To illustrate this, Figure 6 has a comparison of the total execution times for the original and modified versions of our implementation. Again, s-GPU cases were omitted to make the graph more comprehensible. For the same reason, the results for BAS and IDSC descriptors were also not included, as they are similar to the ones observed for ASC.

In a few cases, the effect of the change on the speedup was either neutral (ASC d-CPU/s-CPU) or somewhat positive (d-GPU/s-CPU cases for ASC and CFD). However, we must keep in mind that, since the change reduced the number of operations of both the parallel and serial versions of the algorithm, several speedups obtained with the new implementation were reduced, the most drastic case being d-CPU/s-CPU for SS, which went from $4.2\times$ to $3.8\times$.



Test Case	d-CPU/s-CPU Original	d-CPU/s-CPU Modified	d-GPU/s-CPU Original	d-GPU/s-CPU Modified
Median Speedup	3.3 ×	3.3 ×	3.5 ×	3.6 ×

Figure 6. Comparison of total kernel execution times for different descriptors considering the original and modified algorithms.

Even so, we can look at the modifications in terms of total execution time. With the new implementation, the algorithm became $1.6\times$ faster on average, reaching $1.7\times$ for the SS d-Serial/s-Serial case. The impact of the Sort kernel was also lessened; while it used to be on average 62% of the total execution time, it is now only 36%. The biggest decrease happened in the ASC d-GPU/s-CPU case, where the impact went from 60% to 28%. This way, we see that the modified version of the algorithm is clearly more efficient than the previous one.

6. Conclusions

In this paper, we used OpenCL to accelerate a modified version of the Contextual Spaces Re-Ranking algorithm on the GPU and multi-core CPU of an Accelerated Processing Unit. We analyzed both the effectiveness and efficiency of the proposed approach.

In our efforts to parallelize the original version of the algorithm, we achieved median speedups of up to $3.3\times$ on the implementation of the Compute Distances step. Also, although the sorting algorithm employed for the Re-sort Ranked Lists step was not efficient on GPUs, we obtained median speedups of up to $3.9\times$ when running on the CPU.

The total speedup of the implementation was lessened in many test cases after we modified the algorithm, as the number of operations in both the serial and parallel versions was decreased. Still, the change reduced the total execution time by $1.6\times$ on average.

The modification also improved the effectiveness of the algorithm in most of our test cases. Six out of eleven tests had an increase in the MAP score of at least 1.35% (up to 5.33%). Two of the cases had a decrease in accuracy, but no greater than 0.26%.

Future work includes extending our implementation to run all steps on both the CPU and GPU. Also, we can study the use of different sorting algorithms in order to exploit GPUs and the fact that now only one sort operation with more changes is performed. Furthermore, using larger image collections will enable us to test the scalability of the approach and other possible compromises between effectiveness and efficiency.

Acknowledgments

The authors thank AMD, FAPESP (grant 2013/08645-0), CAPES, and CNPq (grants 306580/2012-8, 484254/2012-0) for their financial support.

References

AMD, A. M. D. I. (2013). AMD Accelerated Parallel Processing OpenCL™ Programming Guide. Accessed: July 22, 2015.

Bentley, J. (2000). *Programming Pearls*. ACM Press Series. Addison-Wesley.

Datta, R., Joshi, D., Li, J., and Wang, J. Z. (2008). Image retrieval: Ideas, influences, and trends of the new age. *ACM Comput. Surv.*, 40:5:1–5:60.

Ferreira, J. R., Oliveira, M. C., and Freitas, A. L. (2014). Performance Evaluation of Medical Image Similarity Analysis in a Heterogeneous Architecture. In *Proc. IEEE CBMS*, pages 159–164.

Latecki, L. J., Lakmper, R., and Eckhardt, U. (2000). Shape Descriptors for Non-rigid Shapes with a Single Closed Contour. In *Proc. CVPR*, pages 424–429.

Pedronette, D. C. G. and da S. Torres, R. (2010). Shape Retrieval using Contour Features and Distance Optmization. In *Proc. VISAPP*, pages 197–202.

Pedronette, D. C. G. and da S. Torres, R. (2011). Exploiting contextual spaces for image re-ranking and rank aggregation. In *Proc. ICMR*, pages 13:1–13:8.

Pedronette, D. C. G., da S. Torres, R., Borin, E., and Breternitz, M. (2012). Efficient Image Re-Ranking Computation on GPUs. In *Proc. ISPA*, pages 95–102.

Pedronette, D. C. G., da S. Torres, R., Borin, E., and Breternitz, M. (2013). Image re-ranking acceleration on gpus. In *Proc. SBAC-PAD*, pages 176–183.

Sevilla, J., Bernabe, S., and Plaza, A. (2014). Unmixing-based content retrieval system for remotely sensed hyperspectral imagery on GPUs. *J. Supercomput.*, 70(2):588–599.

Steele, J. and Cochran, R. (2007). Introduction to GPGPU Programming. In *Proc. ACM-SE 45*, pages 508–508.

Stone, J. E., Gohara, D., and Shi, G. (2010). OpenCL: A Parallel Programming Standard for Heterogeneous Computing Systems. *Comput. Sci. Eng.*, 12:66–73.

Teodoro, G., Valle, E., Mariano, N., Torres, R., Meira Jr, W., and Saltz, J. H. (2014). Approximate similarity search for online multimedia services on distributed CPU–GPU platforms. *VLDB J.*, 23(3):427–448.

Wang, J., Li, Y., Bai, X., Zhang, Y., Wang, C., and Tang, N. (2011). Learning context-sensitive similarity by shortest path propagation. *Pattern Recognit.*, 44(10-11):2367–2374.

Yang, X., Bai, X., Latecki, L. J., and Tu, Z. (2008). Improving Shape Retrieval by Learning Graph Transduction. In *Proc. ECCV*, volume 4, pages 788–801.

Yang, X. and Latecki, L. J. (2011). Affinity learning on a tensor product graph with applications to shape and image retrieval. In *Proc. CVPR*, pages 2369–2376.